



REMARKS

Claims 1-32 are pending in the present application. Reconsideration of the claims is respectfully requested.

I. 35 U.S.C. § 103, Obviousness, Claims 1-3

The examiner has rejected claims 1-3 under 35 U.S.C. § 103 as being unpatentable over *Wiecek* (U.S. Patent No. 5,210,837; hereinafter referred to as "*Wiecek*") in view of *Liedberg* (U.S. Patent No. 5,943,687; hereinafter referred to as "*Liedberg*"). This rejection is respectfully traversed.

In rejecting the claims, the examiner stated:

As per claim 1, *Wiecek* teaches identifying a path within the routine is shown in column 21 line 41-44 (identifying as a path member selected ones of the nodes in the paths between the graph entry nodes and the graph exit nodes in the initial program flow representation") being executed is shown in column 7 line 53-57 ("An arc or an edge from a first node to a second node exists if the second node is a possible destination (or fall-through path) from the first node, as reflected by the order in which the blocks of instructions in the source program are executed"), a plurality of first type instructions are associated with the path is shown in column 2 line 48-56 ("The computer program contains sequentially-ordered blocks of consecutive instructions written in a first computer language, and the initial program flow representation includes a plurality of nodes each representing a different one of the blocks of instructions. The nodes in the initial program flow representation are connected in paths reflective of the possible sequences which the blocks of instructions may be executed"), translating the first type instructions for the path being executed, wherein first type instructions are translated into second type for execution is shown in column 26, line 10-18 ("converting the first program into an initial program flow representation including a plurality of nodes each representing a different one of the blocks of instructions, the nodes being connected in paths reflective of the possible sequences which the blocks of instructions in the program may be executed in accordance with the first computer program, and the connection in any of the paths").

Wiecek does not specifically teach instruction for unexecuted path remain untranslated. However, *Liedberg* teaches unexecuted path in column 7 line 58-62 ("one might perform a more sophisticated analysis of the contents of the instruction buffers in the re-order buffer in order to count only those non-executed instructions"), non executed instructions inherently including untranslated path as claimed.

RECEIVED
DEC 22 1999
TC 2100 MAIL ROOM

It would have been obvious to one of the ordinary skill in the art at the time of invention was made to combine *Wiecek's* translating method with *Liedberg's* non-executed method because one of the ordinary skill in the art would be motivated to increase the processing speed of the computer system.

Office Action dated September 14, 1999, pages 2-3.

The examiner bears the burden of establishing a *prima facie* case of obviousness based on the prior art when rejecting claims under 35 U.S.C. § 103. *In re Fritch*, 972 F.2d 1260, 23 U.S.P.Q.2d 1780 (Fed. Cir. 1992). In the present case, the examiner has failed to establish a *prima facie* case of obviousness based on the prior art. Specifically, the examiner has provided a motivation that is a general overall motivation to increase processing speed of a computer system. This motivation is a general motivation that would make all inventions obvious. It is always desirable to increase the processing speed of a computer system. Such a motivation is a general motivation and not a motivation that one of ordinary skill in the art would use to combine these references in the present case. This desire is insufficient for use as a basis to modify and combine the references as proposed by the examiner.

Further, the examiner has not correctly interpreted the cited references in view of the claimed features. In summary, the presently claimed invention in the depicted example in the specification is directed towards a process used in a data processing system used to compile bytecodes into machine language instructions during the execution of those constructions. Claim 1 reads as follows:

1. A process in a data processing system executing a routine having a plurality of paths, wherein the routine has includes a plurality of first type instructions and wherein the data processing system executes second type instructions, the process comprising:
 - identifying a path within the routine that is being executed, wherein a plurality of first type instructions are associated with the path; and
 - translating the first type instructions for the path being executed, wherein first type instructions are translated into second type instructions for execution by the data processing system, wherein first type instructions for unexecuted paths remain untranslated.

The features of (1) identifying a path that is being executed and (2) translating the first type instructions for the path being executed are features not recognized by the examiner in the comparison of the presently claimed invention to the cited references. Identifying a path

within a routine that is being executed is a step that occurs while the routine is executed. The translating step occurs for the path being executed, which necessarily requires this step to occur during execution of the path.

In contrast, *Wiecek* does not teach identifying a path within a routine that is being executed as recited in the presently claimed invention. The examiner interpreted *Wiecek* as teaching this feature pointing to column 21, lines 41-44, which reads as follows:

identifying as a path member selected ones of the nodes in the paths between the graph entry nodes and the graph exit nodes in the initial program flow representation.

Wiecek, col. 21, lines 41-44. This portion of *Wiecek* merely states that a path member is identified from the paths. The examiner then points to the following section of *Wiecek* for the proposition that this is code that is being executed:

An arc or an edge from a first node to a second node exists if the second node is a possible destination (or fall-through path) from the first node, as reflected by the order in which the blocks of instructions in the source program are executed.

Wiecek, col. 7, lines 53-57. The examiner has combined these two portions of *Wiecek* to identify a path within a routine that is being executed. Such an interpretation or modification of *Wiecek* is improper.

Additionally, the mere fact that a prior art reference can be readily modified does not make the modification obvious unless the prior art suggested the desirability of the modification. *In re Laskowski*, 871 F.2d 115, 10 U.S.P.Q.2d 1397 (Fed. Cir. 1989) and also see *In re Fritch*, 972 F.2d 1260, 23 U.S.P.Q.2d 1780 (Fed. Cir. 1992) and *In re Mills*, 916 F.2d 680, 16 U.S.P.Q.2d 1430 (Fed. Cir. 1993). The examiner may not merely state that the modification would have been obvious to one of ordinary skill in the art without pointing out in the prior art a suggestion of the desirability of the proposed modification.

In this case, no teaching, suggestion, or incentive is present for taking these two teachings within *Wiecek* and modifying them to become a step in which a path is identified within a routine that is being executed. The first portion of *Wiecek* only teaches identifying a path in a set of nodes. The second portion of *Wiecek* cited by the examiner merely states that the path may reflect an order in which blocks of instructions are executed. No teaching, suggestion, or incentive has been pointed out for actually

identifying a path in a routine that is being executed, as recited in claim 1. Specifically, the step of identifying a path in claim 1 occurs while that routine is being executed.

Next, the examiner cites to the following for the proposition that first type instructions are being translated for the path being executed into a second type for execution:

converting the first program into an initial program flow representation including a plurality of nodes each representing a different one of the blocks of instructions, the nodes being connected in paths reflective of the possible sequences which the blocks of instructions in the program may be executed in accordance with the first computer program...

Wiecek, col. 26, lines 10-18. Nowhere does this section of the cited reference teach translating the first type of instructions for the path being executed. Nowhere does the examiner point out any teaching, suggestion, or incentive for performing this translating for the instructions for a path that is being executed. In other words, this reference does not provide for translating instructions for a path during execution of the path. This cited section merely teaches converting the program, but does not provide any teaching, suggestion, or incentive for doing so for a path that is being executed.

The examiner goes on to combine *Wiecek* with *Liedberg*. Such a combination would not occur when one of ordinary skill in the art considers these references as a whole. "It is impermissible within the framework of section 103 to pick and choose from any one reference only so much of it as will support a given position, to the exclusion of other parts necessary to the full appreciation of what such reference fairly suggests to one of ordinary skill in the art." *In re Hedges*, 228 U.S.P.Q. 685, 687 (Fed. Cir. 1986). When *Wiecek* and *Liedberg* are considered as a whole, one of ordinary skill in the art would not be motivated to combine these two references, as suggested by the examiner. In particular, one of ordinary skill in the art would not look to these two references because they are directed towards solving different types of problems. In particular, *Wiecek* is concerned with the following problem:

The present invention relates to the translation of programs between languages, and in particular, to the conversion of a program written in one machine language into a flow graph and then into a program in a different language, preferably a high level language (HLL).

Wiecek, col. 1, lines 9-14.

Converting programs automatically, however, is a difficult task. If all programs had the same type of structure, then conversion of programs might be relatively easy. However, experience has shown that few programs have the same structure and there is no standard structure for programs. Therefore, a general purpose program translation mechanism must be able to accommodate several different program structures.

Wiecek, col. 1, lines 33-40. As can be seen, *Wiecek* is concerned with the problems associated with translating programs between languages. Specifically *Wiecek* is directed towards converting programs from one language to another, not with translating instructions during execution of a program.

Next, *Liedberg* is concerned with the following problem:

The present invention relates to cache memories, and more particularly to strategies for selecting data to be replaced in a cache memory.

Liedberg, col. 1, lines 5-7.

All of the above cache replacement strategies have as a goal a high cache hit ratio, usually defined as the number of times an attempted cache read is successful at obtaining the data from the cache divided by the total number of attempted cache accesses. (A related measure is the cache miss ratio, usually defined as 1-cache hit ratio.) However, these cache replacement strategies are deficient because they fail to take into account the effects of cache misses, which will inevitably occur.

Liedberg, col. 2, lines 37-45. *Liedberg*, as can be seen, is related to cache replacement strategies. The solutions provided by these references are directed towards program translation and cache replacement. In further contrast, the presently claimed invention is directed towards a process for executing a routine, not program translation or cache translation. Therefore, one of ordinary skill in the art would not be motivated to look to *Liedberg* for the features selected by the examiner in *Liedberg* and combine that feature with *Wiecek* to reach the presently claimed invention.

Even assuming, *arguendo*, the examiner is correct in the assertions regarding the teachings of *Liedberg*, one of ordinary skill in the art would not be motivated to combine *Wiecek* with *Liedberg* in the manner recited by the examiner. The examiner goes on to cite *Liedberg* for leaving unexecuted paths untranslated as follows:

In view of the possibility for overestimating priority values under some circumstances, one might perform a more sophisticated analysis of the contents of

the instruction buffers in the re-order buffer in order to count only those non-executed instructions that actually have a dependence on the subject data item.

Liedberg, col. 7, lines 57-62. This portion of *Liedberg* provides no teaching, suggestion, or incentive for having first type instructions for unexecuted paths remaining untranslated. This portion only provides teaching or suggestions regarding analysis of contents of instruction buffers to count non-executed instructions having a dependence on the subject data item. No teaching, suggestion, or incentive is present for having executed instructions of both a first type and a second type in which unexecuted instructions remain in a first type. With the type of instructions being executed by *Liedberg* it is more common to have those instructions already in a machine language form. No teaching, suggestion, or incentive is present for having instructions of two forms. Such a teaching has been selected from *Liedberg* and combined with *Wiecek* without full appreciation of what these references teach.

Moreover, the examiner may not use the claimed invention as an "instruction manual" or "template" to piece together the teachings of the prior art so that the invention is rendered obvious. *In re Fritch*, 972 F.2d 1260, 23 U.S.P.Q.2d 1780 (Fed. Cir. 1992). Such reliance is an impermissible use of hindsight with the benefit of applicant's disclosure. *Id.* Therefore, absent some teaching, suggestion, or incentive in the prior art, *Wiecek* and *Liedberg* cannot be properly combined to form the claimed invention. These two references recognize different problems and provide different solutions for those problems. As mentioned above, *Wiecek* is directed towards program translation problems, while *Liedberg* is directed towards problems associated with cache replacement. The environment in *Wiecek* and *Liedberg* are vastly different from each other and from those of the presently claimed invention. The program translation problems are solved using processes that translate programs from one form to another, while the cache replacement problems are solved using an improved cache replacement technique, including associating priority values with data items. In contrast, the presently claimed invention is directed towards a mechanism for translating or compiling instructions during execution of a routine in which paths that remain unexecuted are untranslated or uncompiled.

In summary, the present invention is directed towards an improved process for executing a routine, not program translation or cache replacement. Thus, when the cited references are considered as a whole by one of ordinary skill in the art, the teachings, suggestions, and incentives provided by these two references would not lead one of ordinary skill in the art to combine these references to reach the presently claimed invention. As a result, absent any teaching, suggestion, or incentive from the prior art to make the proposed combination, the presently claimed invention can be reached only through the an impermissible use of hindsight with the benefit of applicant's disclosure as a model for the needed changes.

Therefore, the rejection of claims 1-3 under 35 U.S.C. § 103 has been overcome.

II. 35 U.S.C. § 103, Obviousness, Claims 4-32

The examiner has rejected claims 4-32 under 35 U.S.C. § 103 as being unpatentable over *Kolawa* et al. (U.S. Patent No. 5,784,553; hereinafter referred to as "*Kolawa*") in view of *Liedberg* (U.S. Patent No. 5,943,687; hereinafter referred to as "*Liedberg*"). This rejection is respectfully traversed.

In rejecting the claims, the examiner stated:

As per claim 4, *Kolawa* teaches identifying a path within the method that is being executed is shown in ABSTRACT line 21-23 ("A dynamic symbolic execution consists of a symbolic execution of the program performed along the path that corresponds to the trial set of actual inputs"), bytecodes are associated with the pats is shown in column 21 line 18-21 ("The SVM symbolically interprets the JAVA program under test using information stored in the JAVA Bytecodes 210. Upon completion, symbolic expressions and associated path conditions are output by the SVM for all required branch conditions (block 302)"), compiling the byte codes is shown in ABSTRACT line 3-5 ("The JAVA program comprises program statements and program variables represented as JAVA source code and compiled by a JAVA compiler into JAVA bytecodes"), byte codes are compiled into native machine code is shown in column 17 line 51-54 ("The bytecodes are a relatively high-level representation of the source code so that some optimization and machine code generation (via a just-in-time compiler 214) can be performed at that level"), *Kolawa* teaches bytecodes remain uncompiled is shown in column 5 line 53-55 ("The original source code 11 comprises all types of files used to express an uncompiled, that is, non-object code, computer program, including definitional and declarative files").

Kolawa does not specifically teach bytecodes for unexecuted paths remain uncompiled. However, *Liedberg* teaches unexecuted path in column 7 line 58-62 ("one might perform a more sophisticated analysis of the contents of the instruction buffers in the re-order buffer in order to count only those non-executed instructions"), non executed instructions inherently including untranslated path as claimed.

It would be obvious to one of ordinary skill in the art at the time of invention was made to combine *Kolawa*'s byte code method with *Liedberg*'s non-executed method because one of the ordinary skill in the art would be motivated to continue compilation of unexecuted paths when these paths are executed for the efficient optimization of the computer system.

Office Action dated September 14, 1999, pages 4-5. In the rejection of these claims, the examiner also has failed to establish a *prima facie* case of obviousness based on the prior art.

Applicant respectfully submits that the references cited cannot be combined to produce the claimed invention. The rule is:

Obviousness cannot be established by combining the teachings of the prior art to produce the claimed invention absent some teaching, suggestion or incentive supporting the combination.

In re Geiger, 815 F.2d 686, 688, 2 U.S.P.Q.2d 1276, 1278 (Fed. Cir. 1987). The incentive or suggestion pointed out by the examiner is not based on the prior art, but is one of a general nature, stating that it would be desirable to have an improved data processing system. Such an incentive does not provide one of ordinary skill in the art any motivation to look to these two references and use the teachings selected by the examiner to reach the presently claimed invention. The examiner has not pointed out any teaching, suggestion, or incentive as to why one of ordinary skill in the art would choose these two references out of all of the references and combine them to reach the presently claimed invention.

A general desire to efficiently optimize a computer system is a general desire expressed by the examiner, which is an insufficient reason for combining references to establish a *prima facie* case of obviousness. The examiner has failed to point out a teaching, suggestion, or incentive to modify or combine the references in the manner proposed by the examiner. Optimization of a computer system is an overall desire present for most inventions involving computers or software. The actual motivations and

teachings of these references would not motivate one of ordinary skill in the art to combine and modify these references. These reasons and motivations are discussed in more detail below.

Claim 4 of the present invention reads as follows:

4. A process in a data processing system for executing a method having a plurality of paths, wherein the data processing system executes native machine code, the process comprising:
 - identifying a path within the method that is being executed, wherein a plurality of bytecodes are associated with the path; and
 - compiling bytecodes for the path being executed, wherein the bytecodes are compiled into native machine code executed by the data processing system, wherein bytecodes for unexecuted paths remain uncompiled.

The examiner has, in this case, incorrectly interpreted the teachings of *Kolawa*. One of ordinary skill in the art would not interpret *Kolawa* as identifying a path within a method that is being executed based on the cited portion of *Kolawa* pointed out by the examiner. This cited portion reads as follows:

A dynamic symbolic execution consists of a symbolic execution of the program performed along the path that corresponds to the trial set of actual inputs.

Kolawa, Abstract, lines 21-23. This portion of *Kolawa* merely provides a definition of dynamic symbolic execution, which consists of a symbolic execution of a program performed along a path that corresponds to particular types of inputs. Nowhere does this portion of *Kolawa* provide any teaching, suggestion, or incentive for identifying a path within a method that is being executed.

In contrast to this teaching, the presently claimed invention specifically identifies a path in a method that is being executed. In other words, this path is identified for a method during execution of the method. In claim 4, the bytecodes for the path being executed are compiled into native machine code, and unexecuted paths remain uncompiled according to the present invention. Such a feature is not suggested by *Kolawa*.

Further, one of ordinary skill in the art would not combine *Kolawa* with *Liedberg* when these references are considered as a whole. As stated above, *Liedberg* is concerned with cache replacement. *Kolawa* is concerned with creating a test suite:

The present invention relates to a method and system for generating a computer program test suite and, in particular, to a method and system for generating a computer program test suite using dynamic symbolic execution for a computer program written in the JAVA programming language.

Kolawa, col. 1, lines 21-25.

Therefore, what is needed is an automated test suite generation tool for assisting a programmer in the generation and the execution validation of test suites capable of covering most source code branches and fully exercising a program's functionality. Desirably, such a tool would automatically generate a test suite for satisfying a required testing criteria, such as statement, branch, segment, error condition, data or total path coverage. Moreover, such a tool would be scalable to operate on programs of all sizes and in particular on programs comprising thousands to millions of lines of source code. In addition, such a tool would automatically generate different forms of input sets, including character and graphical input data. Such a tool would drastically reduce the cost of software development by reducing the man-hours spent on test suite generation. Significantly improve software quality through complete testing of substantially all parts of a program and encourage programmers to test source code at earlier phases of the program development cycle. Additionally, such a tool should be extensible for adaptation to new programming languages and runtime environments such as the JAVA programming language and the JAVA virtual machine.

Kolawa, col. 2, lines 12-32. To solve this problem, *Kolawa* provides the following:

An embodiment of the present invention is a method and system for generating a test suite for a computer program. The computer program comprises program statements and program variables, including at least one input statement having one or more input variables, that are grouped into code blocks and stored in a program database. The test suite comprises sets of inputs. Each of the sets of inputs corresponds to each of the input statements. The program statements corresponding to a candidate code block are read from the program database. Each of the input variables for each input statement and each of the program variables are represented in symbolic form as a symbolic memory value and transforming each program statement dependent on such an input variable into a symbolic expression. A trial set of inputs for each of the input statements is created by finding a solution to the symbolic expression comprising actual input values corresponding to each symbolic memory value using dynamic symbolic execution. An execution run of the computer program is performed using the trial set of inputs and analyzing results obtained from the execution run for coverage of the candidate code block. The trial set of inputs are stored into the test suite if coverage of the candidate code block was obtained.

Kolawa, col. 2, lines 45-67. In contrast, the solution in *Liedberg* is as follows:

The foregoing and other objects are achieved by utilizing a cache data replacement technique in a computer system having a central processing unit (CPU), a cache memory and a main memory, wherein the cache memory has a plurality of data items stored therein. In accordance with one aspect of the invention, the cache data replacement technique includes associating a priority value with each of the stored data items, wherein for each data item, the priority value is an estimate of how much CPU stall time will occur if an attempt is made to retrieve the data item from the cache memory when the data item is not stored in the cache memory.

Liedberg, col. 2, lines 53-64. As can be seen, the solution provided in *Liedberg* is one having to do with cache replacement, while the solution provided in *Kolawa* is one pertaining to a test suite for a computer program. The problems and solutions recited in *Liedberg* and *Kolawa* are the teaching, suggestions, and incentives that one of ordinary skill in the art would look to, not a general desire for optimizing a computer system as stated by the examiner.

Consequently, one of ordinary skill in the art would not be motivated to combine these two references in the manner suggested by the examiner when they are considered as a whole. Further, these problems and solutions also differ from those of the presently claimed invention. No teaching, suggestion, or incentive that would be considered by one of ordinary skill in the art has been pointed out for modifying and combining the teachings in *Kolawa* and *Liedberg* in a process for executing method, as recited in claim 4. In contrast, the presently claimed invention in claim 4 is directed towards a method for selectively compiling bytecodes in which bytecodes for a path being executed are compiled. The result is that bytecodes in unexecuted paths for a method remain uncompiled. As a result, these cited references are directed towards different problems and provide different solutions from each other and from that of the present invention. Therefore, one of ordinary skill in the art would not be motivated to look to these references.

Consequently, in the present case, the selected teachings from each of the references can be chosen only using hindsight with the benefit of applicant's claimed invention to piece together and modify the teachings to reach the presently claimed invention. Such a combination is improper and does not satisfy the burden of establishing a *prima facie* case of obviousness. Again, the general incentive of efficient

optimization of the computer system is a general desirability, not a suggestion or incentive for combining these two particular references. Further, a proper *prima facie* case of obviousness must be supported by some teaching or suggestion contained in the combined references. Such a combination is an improper use of hindsight with the benefit of applicant's invention as a template or road map to piece together and modify the teachings and the prior art. Thus, unless the examiner can provide a teaching, suggestion, or incentive that one of ordinary skill in the art would follow to make the stated combination, these references cannot be combined in the manner necessary to form a *prima facie* case of obviousness for claim 4.

Claims 5-32 are claims depending from claim 4 or independent claims containing similar features and are patentable over the cited references for the same reasons as claim 4. Therefore, the rejection of claims 4-32 under 35 U.S.C. § 103 has been overcome.

III. Conclusion

It is respectfully urged that the subject application is patentable over *Wiecek*, *Liedberg*, and *Kolawa* and is now in condition for allowance.

The examiner is invited to call the undersigned at the below-listed telephone number if in the opinion of the examiner such a telephone conference would expedite or aid the prosecution and examination of this application.

DATE: December 14, 1999

Respectfully submitted,



Duke W. Yee
Reg. No. 34,285
Carstens, Yee & Cahoon, LLP
P.O. Box 802334
Dallas, TX 75380
(972) 367-2001
Attorney for Applicant